

## DOCUMENT RESUME

ED 186 015

IE 008 377

AUTHOR Gentner, Donald R.: And Others  
TITLE COACH: A Schema-Based Tutor.  
INSTITUTION California Univ, San Diego. Center for Human Information Processing.  
SPONS AGENCY Office of Naval Research, Washington, D.C. Personnel and Training Branch.  
REPORT NO UC-TR-7903  
PUB DATE 9 Nov 79  
CONTRACT N00014-76-C-0628: NR-154-387  
NOTE 22p.  
EDRS PRICE MF01/PC01 Plus Postage.  
DESCRIPTORS \*Computer Managed Instruction; \*Computer Oriented Programs; Models: Organization: Programing Languages; \*Simulation: Teaching Models; \*Tutorial Programs  
IDENTIFIERS FLOW

## ABSTRACT

The Coach system, a computer simulation of a human tutor, was constructed with the goal of obtaining a better understanding about how a tutor interprets the student's behavior, diagnoses difficulties and gives advice to a student learning a simple computer programming language. Coach is based on a hierarchy of active schemas representing the tutor's general concepts, more specific information represented in a semantic network, and the coordination of data driven and conceptually guided processing which enable it to interpret behavior, recognize errors, and give advice. This paper describes the Coach system as it gives advice to a student learning FLOW, a simple computer programming language. (Author)

\*\*\*\*\*  
\* Reproductions supplied by EDRS are the best that can be made \*  
\* from the original document. \*  
\*\*\*\*\*

ED186015

U.S. DEPARTMENT OF HEALTH,  
EDUCATION & WELFARE  
NATIONAL INSTITUTE OF  
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY.

**Coach  
A Schema-Based Tutor**

**Donald R. Gentner**

*Center for Human Information Processing  
University of California, San Diego*

The work reported in this paper was carried out in collaboration with Donald A. Norman, David E. Rumelhart, and Mark R. Wallen. This research was supported by the Office of Naval Research, Personnel and Training Research Programs and the Advanced Research Projects Agency, and was monitored by ONR under Contract N00014-76-C-0628, NR 154-387, under terms of ARPA Order No. 2284.

IR008377

## COACH: A SCHEMA-BASED TUTOR

### Abstract

*The Coach system, a computer simulation of a human tutor, was constructed with the goal of obtaining of a better understanding of how a tutor interprets the student's behavior, diagnoses difficulties, and gives advice. Coach gives advice to a student who is learning a simple computer programming language. Its intelligence is based on a hierarchy of active schemas which represent the tutor's general concepts, and on more specific information represented in a semantic network. The coordination of conceptually-guided and data-driven processing enables the Coach system to interpret student behavior, recognize errors, give advice to the student.*

### Introduction

This paper describes the Coach system, an intelligent computer-based instructional system which uses a schema representation of knowledge to interpret student actions and to give advice to the student. The Coach system gives advice to a student learning FLOW, a simple computer programming language. FLOW is similar to BASIC, with about fifteen different statements, three commands, and simple debugging aids. It is based on a language originally developed by Raskin (1974) as an introductory computer language. Earlier versions of the FLOW tutor used declarative representations of schemas (Gentner & Norman, 1977; Gentner, 1979), in contrast with the present Coach system where schemas are active processes.

The Coach system falls within the tradition of intelligent computer-assisted instructional systems originated by Carbonell (1970) and continued by such workers as Burton and Brown, Miller, and Goldstein (see the collection edited by Sleeman and Brown, 1979). The basic principles of the Coach system can be summarized briefly:

- a) The Coach system is based on a set of active processes, called schemas, which interpret the student's behavior, detect errors, and generate advice to the student. The schemas form a nested hierarchy to represent generic concepts such as programming constructs, the structure of written text, teaching strategies, common errors, and individual keypresses.
- b) Specific information, such as the instructional manual, the structure of the FLOW language, and the tutor's current model of the student, is represented in a semantic network.
- c) Conceptually-guided processing. The tutor expands high-level schemas into their component lower-level schemas to predict and observe student behavior, solve programming problems, and give advice. When schemas are unable to find the predicted student behavior, they may suspend processing, but keep the entire structure of predictions intact for possible reactivation later.
- d) Data-driven processing. Unexpected student actions activate low-level schemas which attempt to assemble themselves into structures of schemas representing errors or alternate correct actions.
- e) Conceptually-guided and data-driven processes may interact in several ways. Higher-level schemas can incorporate already existing structures of schemas which have been generating by data-driven processing. Lower-level schemas can directly or indirectly activate high-level schemas which they hope will eventually incorporate them by conceptually-guided processing.
- f) The sequence of processing is primarily controlled by the individual schemas which activate related higher or lower level schemas. There is also a global agenda of schemas waiting to be activated, typically containing one to three high-level schemas.

### Basic structure and operation of schemas

The Coach system is implemented in Micro-Sol, a language which constructs, modifies and interprets active semantic network structures. Micro-Sol is based on Sol (Norman, Rumelhart & the LNR Research Group, 1975), but lacks the natural language capabilities of Sol. Micro-Sol currently runs on a PDP 11/45 minicomputer.

Schemas in the Coach system are used to represent concepts at many levels. At the highest level there are schemas such as those for instructional manuals and the functions of programs. There are schemas for program constructs such as loops, for common student errors, and for the statements in the FLOW language. At the lowest level there are the schemas for the individual keypresses and periods of student inactivity. Schemas have arguments, which serve to distinguish the various instances of a given schema. I have used a family analogy to describe the hierarchically nested schemas: the sub-parts of a schema are referred to as the children of the schema; the schema is itself normally part of a higher-level schema known as the parent.

A schema in the Coach system corresponds to a restricted type of procedural definition in Micro-Sol. Figure 1 shows the general structure of a schema definition. When a schema becomes active, it searches for its component children by activating the corresponding schemas. If a child is absent, the schema normally suspends operation and returns to its parent with the truth value "maybe", indicating that it was unable to complete itself. If a suspended schema is later reactivated, the schema starts up again at the place where it suspended. When all its children have been found, the schema may perform some actions, such as updating the model of the student. Next the schema determines whether it was originally invoked by a higher-level parent schema. If not, i.e. when the schema is an orphan, the schema suggests some possible parents by putting their schemas on an agenda. Instead of merely suggesting possible parents, an orphaned schema which is more confident of its function may directly activate a potential parent. Finally, the schema returns to the procedure which originally activated it with a truth value of "true", indicating that the schema is complete. In the normal case where the schema has a parent, it would be returning to its parent. If a schema is completely unable to find its children, it can return to its parent with a truth value of "false".

The collection of schemas makes up a distributed intelligence system, without an overall executive procedure. The highest level process in the Coach system simply activates the next schema on the agenda. Information flow is primarily restricted to communication from parent to child (via the arguments used when invoking the child's schema) and from child to parent (a child returns with a truth value indicating whether or not it is complete). In addition, information represented in the semantic network is available to any schema. Many of the schemas maintain or have access to the model of the student, which allows them to coordinate a changing strategy as the student progresses thru the learning task. The Coach system also includes a FLOW simulator which maintains a semantic network representation of the student's terminal screen, current program, and command state.

```
define schema-1 (argument-1 argument-2)
  if 1st child is absent, suspend here and return (maybe)
  if 2nd child is absent, suspend here and return (maybe)
  if 3rd child is absent, suspend here and return (maybe)
  perform actions
  if orphaned,
    suggest possible-parent-1
    suggest possible-parent-2
  return (true).
```

Figure 1. The general structure of a schema.

### Schema-based interpretation

#### *The tutorial environment*

To illustrate the operation of the schemas, I will give a series of examples showing how the Coach system analyzes a student's behavior, but first I need to describe the general instructional setup.

Figure 2 shows a typical experimental arrangement, in this case with a human tutor. On the right, a student is shown in top view, sitting at a terminal that is connected to a minicomputer where the student can enter and execute FLOW computer programs. The student has an instruction booklet that contains a general discussion of computers and computer languages, a description of the FLOW language, examples to try, and problems to solve. The minicomputer executes the student's FLOW programs and thus provides the student with feedback. The system also provides a line-at-a-time execution mode to help the student debug programs. The instructional booklet is fairly complete, and in principle the student could go through the whole booklet without help. Most students, however, have considerable difficulty in completing the instructional sequence without help. In the arrangement shown in Figure 2 there is a tutor in another room who sits in front of a CRT terminal which displays a copy of everything appearing on the student's terminal. The student and tutor can send messages to each other on a pair of interconnected terminals. There are no restrictions on the messages from the tutor, but the student's messages to the tutor are limited to "yes", "no", "ok", and "help".

We record the student's keypresses and the corresponding times for later use. Figure 3 shows an annotated portion of a student record, along with advice produced by the Coach system. The left column gives the time in seconds since the beginning of the session; the student's keypresses are to the right. When the student does not press a key for 11 seconds, the minicomputer indicates a quiet period. Although this record seems rather cryptic at first, human tutors experienced in teaching FLOW can use it to interpret student behavior almost as easily as they can interpret a copy of the student's terminal screen. The task of the Coach system is to interpret this record of keypresses in terms of the student's progress thru the instructional sequence and to give the student appropriate advice when needed. Our primary concern in this research is to learn what general knowledge, specific information, and processing capabilities a tutor must have to perform this task.

#### *Conceptually-guided prediction*

To illustrate how Coach uses conceptually-guided processing to predict and interpret student actions, I present an example in some detail, showing Coach's interpretation of the portion of the student protocol shown in Figure 3.

The example begins with Figure 4, which shows a trace of Coach's processing starting just after the student has pressed the R key to run the current program. That action has completed problem-8 and paragraph-13 in the FLOW instruction manual. The top level schema, *manual*, is now active. (The names of schemas are italicized.) The *manual* schema has a single argument, the title of the manual (in this case "The FLOW Manual"). It expects the student to complete a series of paragraphs in the instructional manual. The *manual* schema examines the node for "The FLOW Manual" in the semantic network to get the names of the paragraphs, and then activates the *paragraph* schema with the name of the next paragraph, paragraph-14, as its argument. This action is reflected in line 2 of the Coach trace in Figure 4. Figure 5 shows the definition of the *paragraph* schema, a typical schema. After first checking to see whether it contains a programming problem in addition to the text, the *paragraph* schema expects the student to complete the text and then updates the reading rate in its model of the student. Next it expects the student to complete the problem, if present. Finally, when all its children are complete, it normally returns to its parent, the *manual* schema. If it is orphaned, however, it will activate

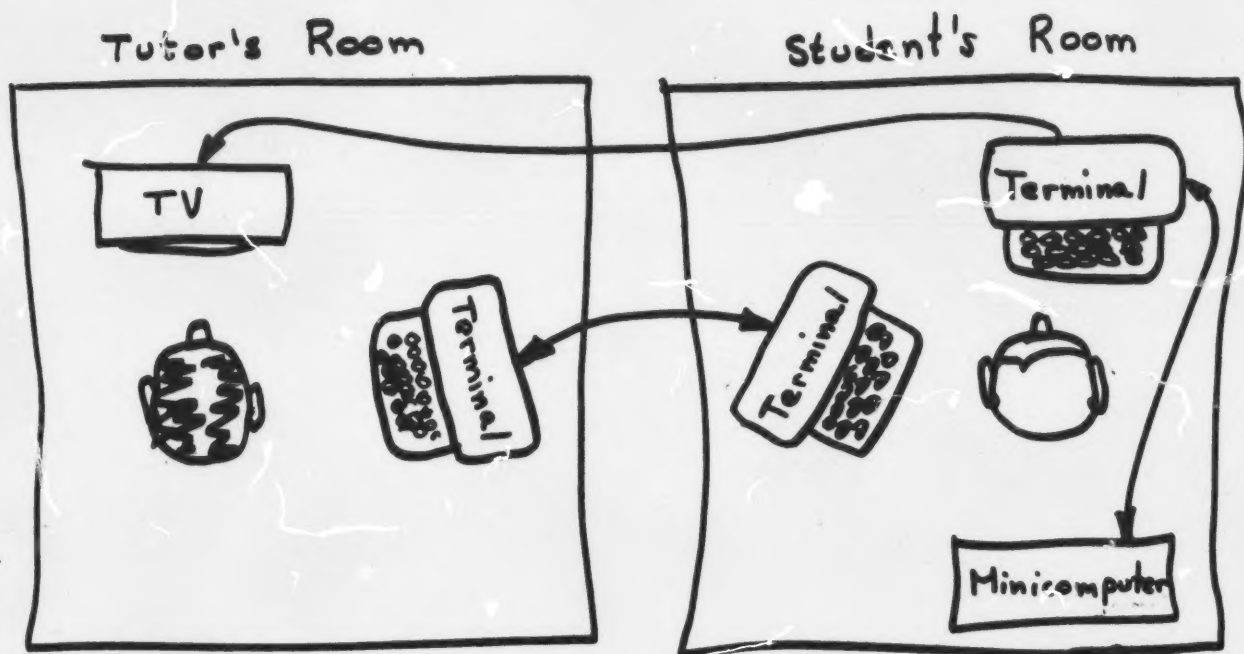


Figure 2. The FLOW instructional setting.

--section omitted--  
00691 quiet  
00702 quiet  
00713 quiet (Student is reading instruction manual.)  
--section omitted--  
00922 quiet  
00933 quiet  
00944 quiet  
00952 RUBOUT (Student presses incorrect keys.)  
00954 D  
00965 quiet  
00976 quiet  
00987 quiet  
00998 quiet  
TUTOR: I think you are trying to modify your program  
You should list your program first.  
01003 RUBOUT (Student presses additional incorrect  
01010 R keys. Except for "R", they have no  
01021 quiet effect, since they are illegal in this  
01022 RUBOUT context.)  
01033 X  
01035 D  
01046 quiet  
01057 quiet  
TUTOR: I was expecting you to list your program  
You should press the L key.  
01069 L  
01079 quiet  
01090 quiet  
01094 RUBOUT (Student goes on to solve the problem.)  
01099 RUBOUT  
01105 1  
01107 5  
01113 SPACE (The "SPACE" and "D" keypresses are  
01124 quiet incorrect, but Coach does not give advice  
01127 D since the student does not pause long  
01138 SPACE enough.)  
01139 quiet  
01150 quiet  
01161 quiet  
01170 SPACE  
01175 QUOTE  
01177 SPACE  
01178 QUOTE

Figure 3. Partial student record. The time in seconds since the beginning of the session and the corresponding keypresses are shown. Advice was generated by the Coach system.

```
1      --section omitted--
2      Expecting paragraph paragraph-14
3      Expecting text text-14 true
4      Expecting read ( 41 ) true
5      Expecting pause ( 61 ) ( 246 )
6      Expecting quiet nil
7      OBSERVING          00691 quiet.
8      Student completed quiet ( 691 )
9      Expecting quiet nil
10     OBSERVING          00702 quiet.
11     Student completed quiet ( 702 )
12     --section omitted--
13     Expecting quiet nil
14     OBSERVING          00933 quiet.
15     Student completed quiet ( 933 )
16     Student is still pausing after 253 seconds.
17     Student completed pause ( 61 ) ( 246 )
18     Student completed read ( 41 ) true
19     Student read about replace
20     Student read about delete
21     Student read about insertt
22     Student read about modify
23     Student completed text text-14 true
24     Student's current reading rate is 4 sec/line.
25     Expecting problem problem-9
26     Expecting modify #02512
27     *02512
28         isainverse #02536      [ display MARY ]
29         isainverse #02538      [ display SPACE ]
30         isainverse #02540      [ display SMITH ]
31     Expecting List
32     Expecting L nil
33     OBSERVING          00944 quiet.
```

Figure 4. An excerpt of the Coach trace (Part1).

```
define "paragraph" (:name)
  {}
  if (firstnode (from :name via problemname)
      &(
        call (the firstnode :cproblem)
        call (true :problem-flag)
      ))
    firstnode (from :name via textname)
    if (absent (text (the firstnode :problem-flag) for token)
        suspend ("3" newnode the newnode maybe))
    update-reading-rate
    if (:problem-flag
        if (absent (problem (:cproblem) for token)
            suspend ("5" newnode the newnode maybe)))
    if (orphaned (token)
        &(
          call (firstnode (from :name via contained-in) :man-name)
          push (manual (:man-name))
        ))
    return (token true).
```

Figure 5. The schema for *paragraph*.

the *manual* schema directly. In the example shown in Figure 4, as the schema *paragraph* (paragraph-14) is activated, it examines the semantic network representation of the The FLOW Manual and finds that paragraph-14 contains problem-9 and text-14, and then activates the *text* schema with the argument "text-14" as reflected in line 3 of the trace. (The second argument of *text*, "true", indicates that the paragraph also contains a problem.)

In a similar manner, the *text* schema determines from the semantic network that text-14 contains 41 lines of text, and activates the *read* schema with 41 as its argument. *Read* then checks the student model to find that student's current reading rate is 3 seconds per line, and activates the *pause* schema, expecting a pause of between 61 and 246 seconds (half and twice the estimated time). Finally the *pause* schema activates a *quiet* schema. Like the schemas for keypresses, the *quiet* schema is a bottom-level schema and it checks to see if the student has actually been quiet for an 11 second period. In this case, there was a quiet period ending at time 00691; the *quiet* schema is completed, and it returns to *pause*, its parent schema.

Before continuing, let us review briefly what has happened. Coach has used the schemas, the semantic network representation of the instructional manual, and the model of the student to predict a period of quiet while the student is reading the text in paragraph-14. When that quiet is observed, Coach assumes that the student is reading. In another context, a period of quiet could mean that the student having difficulty or working on a problem, but conceptually-guided processing has allowed Coach to interpret what otherwise would be an ambiguous event.

When the *pause* schema detects the completed *quiet* schema at line 8 of Figure 4, it determines that the maximum expected pause length has not yet occurred and expects another period of quiet. This sequence repeats a number of times as the student continues to pause. Normally the *pause* schema would terminate when the student pressed a key, but in this case the pause runs to the full length. The *pause* and *read* schemas then return, complete. The *text* schema next examines the representation of the FLOW manual to determine what topics were covered in text-14, updates the model of the student accordingly, and returns (lines 19-23). Finally, the *paragraph* schema updates the student's reading rate and then expects the student to do problem-9.

#### *Data-driven invocation*

As long as the student's actions match Coach's expectations, conceptually-guided prediction provides a natural and efficient means of interpreting the student's keypresses. Often the student's actions do not match predictions, however, and the tutor needs a way of responding to these unexpected events. Coach uses data-driven invocation of schemas to initiate interpretation of unpredicted student actions. There are two modes of invocation: suggesting and pushing.

Figure 6 shows an example of a data-driven suggestion. This section of the trace follows immediately after that shown in Figure 4. Up until this point, the student's actions have matched Coach's expectations and the interpretation has proceeded smoothly. On line 1 of Figure 6, however, Coach is expecting an *L* and instead there is another period of quiet at time 00944. A *quiet* schema is activated, and it suggests (puts on the agenda) a schema for *unexpected-pause*. The *L* schema and all the unsatisfied parents above it suspend themselves, and the next item on the agenda, the *unexpected-pause* schema, is activated. *Unexpected-pause* preempts the orphaned *quiet* schema (line 13) and expects further quiet. (If a 60 second pause is observed, *unexpected-pause* will initiate advice to the student.) Instead of more quiet, however, the student presses the RUBOUT key. The *unexpected-pause* schema, having failed to observe an additional period of quiet, reports the 19 second pause which it did find, and then terminates.

```
1  OBSERVING          00944 quiet.
2  quiet ( 944 ) suggests unexpected-pause
3  L suspending
4  List suspending
5  modify suspending
6  problem suspending
7  paragraph suspending
8  Current agenda
9  agenda
10         unexpected-pause      *02526
11         manual manual-02575
12 Student is pausing unexpectedly.
13 unexpected-pause preempts quiet ( 944 )
14 Expecting quiet nil
15 OBSERVING          00952 RUBOUT.
16 illegal key        RUBOUT ( 952 )
17 RUBOUT ( 952 ) suggests out-of-order *02581
18 Did not observe quiet nil
19 Student unexpectedly paused for 19 seconds.
20 Current agenda
21 agenda
22         manual manual-02575
23         out-of-order      *02580
24 Reactivating suspended schema  paragraph-02575
25 Reactivating suspended schema  problem-02575
26 Reactivating suspended schema  modify-02575
27 Reactivating suspended schema  List-02575
28 Reactivating suspended schema  L-02575
29 OBSERVING          00954 D.
30 illegal key        D ( 954 )
31 D ( 954 ) suggests out-of-order *02593
32 L suspending
33 List suspending
34 modify : suspending
35 problem suspending
36 paragraph suspending
```

Figure 6. An excerpt of the Coach trace (Part2).

In addition to suggesting possible parents by placing them on the agenda, schemas can directly activate a parent schema. This second type of data-driven invocation is called "pushing", and allows a low-level schema to directly control the flow of processing. Note that an orphaned schema cannot directly link itself to a parent schema; an orphan can only suggest or push a parent, which then when has the option of accepting or rejecting the orphaned schema.

#### *Suspension and re-activation of schemas*

In a system such as Coach, which interprets information serially, expectations that are not met at first may be fulfilled later. We need a mechanism to terminate a chain of reasoning if it runs into difficulty, with the possibility of reactivating it later when it might be more successful. In the meantime, processing should focus on the unexpected current events.

Corresponding to the ability of human tutors to hold a set of expectations temporarily in abeyance, schemas in the Coach system can suspend themselves if they fail to find a child and allow processing to proceed on other topics. Schemas can suspend whenever they fail to find an expected child. If they are reactivated later, they resume processing where they suspended and search again for the missing child.

Two examples of the suspension and reactivation of schemas are shown in the Coach trace. The first example is in Figure 6. At the beginning of this section of the trace, Coach had developed a line of conceptually-guided prediction culminating in the *L* schema, which then observed the student. Instead of an *L* keypress, however, the *L* schema found a quiet period (line 2). It suspended itself and returned with the truth value of "maybe" to the *List* schema which had activated it. The *List* schema was thus incomplete, and suspended itself in turn. In a similar fashion the entire predicted chain of schemas suspends itself, until finally the top-level schema, *manual*, suspends and places itself on the global agenda. Only the top-level schema in a chain places itself on the agenda; the other suspended schemas remain linked to their respective parents.

After the *unexpected-pause* schema is activated and terminates (lines 12-19), the suspended *manual* schema comes up on the agenda and is reactivated. Reactivated schemas start out processing at the point where they had suspended. In this case the reactivated *manual* schema starts by looking for paragraph-14 (rather than paragraph-1, as a freshly activated *manual* schema would do), reactivating the suspended *paragraph* schema on line 24.

The *paragraph* schema suspended when it was unable to find a completed schema for problem-9, and it starts by reactivating the schema for problem-9. In a similar way the remaining schemas in the chain are reactivated, until finally the *L* schema observes the student. The *L* schema finds that the student has pressed the D key, and the entire sequence of schemas, from *L* thru *manual*, suspends once again (lines 32-36).

This sequence of developing a line of expectations, suspending it to attend to unexpected data, and reactivating the intact line of expectations later, might be better simulated with a parallel processing implementation of the tutor, but the suspension and reactivation of schemas gives us many of the essential features of multiple lines of interpretation within a serial processing implementation.

#### *Solving problems*

In addition to enabling Coach to follow the student's progress thru the instructional manual, another important use of conceptually-guided prediction is in the solution of FLOW programming problems. Coach interprets the student's solutions to programming problems by "solving" the problem for itself and predicting that the student will solve the problem in the same manner. The problems in the

FLOW Instructional Manual are simple enough that Coach, starting with a functional description of the computer program needed, can effectively write the program by instantiating the schemas for the functions involved. This will be illustrated by examining how Coach solves problem-9.

In problem-8, the student ran the program shown in Figure 7. The output which this program produced is shown in the lower part of the figure. (The student's name has been changed.) After describing various ways to modify a program, the instruction manual states problem-9:

Choose one (or all) of these methods and modify your program so that it displays your name with a space between your first and last names.

Coach starts with a corresponding representation of the problem in its semantic network:

modify to set-of (display (firstname) display ("SPACE") display (lastname)).

On line 25 of Figure 4, after concluding that the student has completed text-14, a schema for problem-9 is activated. The *problem* schema finds the description of problem-9 in the semantic network, and activates the *modify* schema. The argument of the *modify* schema, shown on lines 27 - 30, is essentially a functional description of the required program. The proper values for firstname and lastname were obtained from the model of the student as Coach instantiated the *modify* schema. According to the definition of the *modify* schema, before a program can be modified, it must be displayed on the screen using the LIST command. Since the screen now contains the output from the previous run, Coach predicts that the student will press the L key to list the program. After the student finally lists the program, Coach continues with the solution to problem-9 in a section of the trace not shown here. The *modify* schema calculates the function of the student's current program, compares that to the desired program function, and finds that a display SPACE function must be inserted after line 10. Eventually, Coach deduces that the statement number must be changed to a number between 11 and 19, which requires the student to press the RUBOUT key. Later Coach continues the solution of the problem, expanding the schema for displaying a SPACE into a schema for a display-quoted-string statement, and finally a D keypress. After a few false starts, the student presses the D key, and Coach goes on to predict a quoted string consisting of the key sequence QUOTE, SPACE, and QUOTE.

Thus the unfolding definitions of schemas allow the Coach system to predict how the student will solve the problem. Of course even these simple problems have more than one correct solution. If the student chooses to solve the problem in a different manner, Coach has to use data-driven processing to build up a schema giving a functional description of the student's program, which can then match a predicted schema representing the intended function of the program.

### *Error schemas*

A major function of data-driven processing in the Coach system is the recognition of errors. Coach is intended to simulate an experienced tutor and includes schemas which represent the common types of errors which students make. Schemas invoked by data-driven processing can suggest or push any schemas, including these error schemas. When they are activated, the error schemas search for their children in the normal manner. Successful completion of an error schema corresponds to the recognition of a student error. Once an error has been recognized, Coach normally does not give immediate advice, but rather allows the students to try to recover from the error on their own, and then gives advice when the students pause.

```
010 DISPLAY "MARY"  
020 DISPLAY "SMITH"  
030
```

MARYSMITH

Figure 7. The student's current program and the resulting output.

The Coach trace shows a good example of error interpretation. On line 15 of Figure 6, when Coach was expecting a quiet period, the student pressed the RUBOUT key. The student has just run the current program and the RUBOUT key is illegal in this context.<sup>1</sup> The RUBOUT schema, like the schemas for all other illegal keys, suggests an *out-of-order* schema (with the RUBOUT schema as its argument), which will eventually try to determine if the RUBOUT key was part of a predicted schema but out of order. The next item on the agenda, however, is the suspended *manual* schema, which reactivates all its children and eventually leads to the observation of a D keypress on line 29. The D key is also illegal, and the D schema suggests another *out-of-order* schema.

After the active schemas suspend, the next item on the agenda, at the top of Figure 8, is the *out-of-order* (RUBOUT) schema. *Out-of-order* waits for the student to pause 40 seconds before doing any investigation. In the example shown here, the student paused for 44 seconds (line 16), so *out-of-order* tries to determine if the RUBOUT could be a part of a predicted schema. It makes up a target set consisting of all the currently unsatisfied schemas: schemas lacking either parents or children. *Out-of-order* then refers to the semantic network representation of information about the FLOW language and, starting at RUBOUT, does a breadth-first search along the relation "part-of" looking for a schema in the target set. It eventually finds on line 34 that RUBOUT could be part of the expected *modify* schema, thereby concluding that the student was trying to modify the program, but neglected to LIST the program first.

#### *Giving advice*

Once the Coach system has recognized an error, it can give advice to the student. Three important issues come into focus here: 1) Should the advice be given immediately or should students be allowed an opportunity to detect and correct errors on their own? 2) At what level should the advice be phrased (e.g. Should the advice be in terms of program functions or individual keypresses)? 3) What should be done if the student does not respond to the advice?

Coach decides when to give advice based on its model of the student. Advice is given relatively soon when the student is unfamiliar with the concept at issue. When the student has successfully used the relevant concept earlier, advice is delayed to allow the student to debug the program without help. Each schema in the Coach system corresponds to a concept in the FLOW instructional course. Coach maintains a semantic network representation of the student's experience with each of the schemas. Originally schemas are not associated with the student model. When the student reads about a concept in the instructional manual, Coach notes in the student model that that student "read about" the corresponding schema. When the student successfully completes the schema for the first time, it is noted as "used", and when the student completes the schema a second time Coach notes that the student "mastered" the schema.

We can see how this information in the student model is used by continuing the analysis of the "out of order" problem. On line 34 of Figure 8 Coach has inferred that the RUBOUT key was part of an attempt to modify the program, but was out of order. The advice will therefore be given at the level of the *modify* schema. As indicated in the next line, Coach finds from the model of the student that the student has read about modifying programs, but has never done any modification. Based on this information, the *out-of-order* schema decides to give advice immediately. (It had already allowed a 40 second pause before attempting to interpret the RUBOUT key.) Coach would have waited for an

-----

1. Only the R, W, and L keys, for the RUN, WALK, and LIST commands are legal at this point. Syntactically illegal keys have no effect in the FLOW system. The illegal key is displayed briefly on the terminal screen and then erased as the terminal beeps.

```
1      Current agenda
2      agenda
3          out-of-order      *02580
4          out-of-order      *02592
5          manual    manual-02605
6      Out-of-order ( RUBOUT ) waiting for 40 seconds
7      Expecting pause ( 40 ) ( 40 )
8      Expecting quiet nil
9      OBSERVING      00965 quiet.
10     Student completed quiet ( 965 )
11     --section omitted--
12     Expecting quiet nil
13     OBSERVING      00998 quiet.
14     Student completed quiet ( 998 )
15     Student is still pausing after 44 seconds.
16     Student completed pause ( 40 ) ( 40 )
17     targetset is
18     *02627
19         isainverse      unexpected-pause
20         isainverse      L
21         isainverse      List
22         isainverse      modify
23         isainverse      problem
24         isainverse      paragraph
25         isainverse      manual
26     bfsearch starting at RUBOUT
27     searching along part-of
28     bfsearch checking character-delete
29     bfsearch checking change-statement-number
30     bfsearch checking insert
31     bfsearch checking delete
32     bfsearch checking append
33     bfsearch checking modify
34     bfsearch found modify
35     Out-of-order found student read about modify
36     Therefore will give help immediately
37     TUTOR      I think you are trying to modify your program
38               You should list your program first.
```

Figure 8. An excerpt of the Coach trace (Part 3).

additional 20 second pause before giving advice if the student had actually used the *modify* schema successfully on some previous occasion, or waited for a 40 second pause if the student had mastered the *modify* schema. The teaching strategy here is to give students a chance to work out their own problems if they seem to know the required concepts, but to give help early when they are dealing with new concepts.

The actual advice which Coach gives is generated fairly simply. *Out-of-order* has assumed that the problem is with the *modify* schema. It examines the suspended *modify* schema and finds that *modify* is currently searching for a *List* schema. *Out-of-order* has an advice frame which looks like:

"I think you are trying to <schema>.

You should <current child of schema> first."

The bracketed arguments in the advice frame are filled in with phrases associated with the corresponding schemas, and Coach produces the advice shown in lines 37 and 38.

Whenever Coach gives advice to perform a specific task, in this case to list the program, it invokes a *monitor* to ensure that the student performs that task correctly. In this case, the *monitor* checks the model of the student, finds that the student has used the *LIST* command once, and decides to allow a 20 second pause before giving further advice. *Monitor* essentially narrows Coach's focus of attention. As long as the student does not change the state of the *FLOW* system, Coach looks only for completion of the *List* schema or the 20 second pause. Other non-critical information is ignored. The level of Coach's attention has also changed. Coach originally thought the problem was at the level of the *modify* schema, and advised the student to list the program. With this particular student, however, that advice was not effective because the student had forgotten how to list the program. As we have seen, Coach's attention has now shifted to the *List* schema, and subsequent advice at that level led the student out of difficulty.

### *Finding children*

So far, I have not described in detail how a schema looks for its children. In most of the cases we've examined, the parent schema instantiates a new schema for its children and activates the schema. But activation of a new schema is the last resort after the parent has made three other attempts to find its child. In order to take advantage of data-driven and suspended processing, the parent schema must first check to see if a suitable child already exists before activating an entirely new schema.

Schemas search for their children using the "absent" procedure, which instantiates a schema for the child and then tries to find an available schema which "matches" the new instance. A schema is said to match the instance if it has the same name, and the same or more constrained arguments. The search for a suitable child proceeds in four steps.

First, the "absent" procedure checks for a matching orphan (i.e. a schema without a parent). For example, on line 13 of Figure 6, the *unexpected-pause* schema finds an orphaned *quiet* schema which it preempts. Preempting an orphaned schema can have side effects, if that orphan has suggested other schemas. On lines 16 and 17 in Figure 9 an unexpected *D* schema suggested schemas for two different *FLOW* statements which it could be part of (display-quoted-string and display-variable). Later on line 33, when another *display-quoted-string* schema, (which had been a conceptually-guided prediction), preempts the *D* schema, the support for the two suggested schemas collapses and they are removed from the agenda. This is consistent with the general rule that a schema may have only one parent at a time.

Second, if an appropriate orphan is not found, the parent schema is checked to see if it has previously activated and suspended this child. If so, the suspended schema is reactivated. There are many

```

1      --section omitted--
2      Student completed change-statement-number ( 11 ) ( 19 )
3      Expecting display SPACE
4      Expecting display-quoted-string anything SPACE
5      Expecting D nil
6      OBSERVING      01113 SPACE.
7      --section omitted--
8      Expecting quiet nil
9      OBSERVING      01127 D.
10     currentline
11         contains *00602      [ 0 nil ]
12         contains *02802      [ 1 ( 1105 ) ]
13         contains *02808      [ 5 ( 1107 ) ]
14         contains *02853      [ D ( 1127 ) ]
15     The current state is now DISPLAY
16     D ( 1127 ) suggests display-quoted-string nil
17     D ( 1127 ) suggests display-variable nil
18     Did not observe quiet nil
19     Student paused for 14 seconds.
20     Did not observe pause ( 40 ) ( 40 )
21     Current agenda
22     agenda
23         manual manual-02841
24         display-quoted-string      *02854
25         display-variable            *02855
26     Reactivating suspended schema paragraph-02840
27     Reactivating suspended schema problem-02838
28     Reactivating suspended schema modify-02836
29     Reactivating suspended schema insertf-02834
30     Reactivating suspended schema insertt-02832
31     Reactivating suspended schema display-02830
32     Reactivating suspended schema display-quoted-string-02828
33     display-quoted-string ( 15 ) SPACE preempts D ( 1127 )
34     display-variable nil collapses.
35     display-quoted-string nil collapses.
36     Expecting qstring SPACE
37     Expecting QUOTE nil
38     OBSERVING      01128 SPACE.
39     --section omitted--
40     Student completed QUOTE ( 1175 )
41     Expecting SPACE anything
42     OBSERVING      01177 SPACE.
43     currentline
44         contains *00602      [ 0 nil ]
45         contains *02802      [ 1 ( 1105 ) ]
46         contains *02808      [ 5 ( 1107 ) ]
47         contains *02853      [ D ( 1127 ) ]
48         contains *02871      [ QUOTE ( 1175 ) ]
49         contains *02944      [ SPACE ( 1177 ) ]
50     Student has used SPACE once.
51     Student completed SPACE ( 1177 )
52     Expecting QUOTE nil
53     OBSERVING      01178 QUOTE.

```

Figure 9. An excerpt of the Coach trace (Part 4).

instances of the reactivation of a child in the Coach trace. For example in lines 24 - 28 of Figure 6, a series of suspended schemas are reactivated and reactivate their children in turn. A schema has access only to suspended schemas for children which it originally activated.

Third, if there was no suspended child, "absent" looks for a matching schema which has been suggested and placed on the global agenda. If found, the schema is taken off the agenda and activated directly rather than having to wait its turn. This happens relatively infrequently, and there are no examples of "taking a suggestion" in this section of the Coach trace.

Fourth, the newly instantiated schema itself is activated. This is the normal case as conceptually-guided predictions are generated, for example in lines 2 - 6 of Figure 4. In all cases the new child is linked to the parent schema. The child will be unlinked later if it does not successfully complete itself.

### Discussion

The original intent in building the Coach system was to explore how a human tutor interprets the student's behavior and gives advice, rather than to build a complete instructional system. In terms of a working CAI system, there are a number of limitations in the present implementation of the Coach system. The schemas in the current system cover only the material presented in the first half of the FLOW instruction manual (about 30 minutes of instructional time.) In addition, the current set of error schemas are sufficient for only a portion of the errors which we have seen in students. These limitations primarily reflect the fact that Coach is currently implemented on a minicomputer with limited memory size. There is no reason in principle that the number of schemas could not be enlarged to handle a wider range of material. At this point it is not clear what the ultimate limits of this approach are. A more serious source of limitations is that Coach predicts the student's solution of programming problems by solving the problems itself. With more complex programming problems, this would not be possible; we would be faced with all the difficulties of automatic programming.

The functioning of schemas in the Coach system does not properly simulate our models of human thought in some areas. For instance, only one schema is active at a time. A more attractive model of human perception is one in which many schemas are simultaneously active, competing for the data and substructures, and interacting with each other to give alternate perspectives on the environment. Some of the assumptions and restrictions built into the Coach system are based more on intuition than psychological theory. For example, in the Coach system, schemas can have only one parent at a given time. This was intended to correspond to the idea that we cannot interpret data (such as a Necker cube) from two different perspectives simultaneously. But one could also argue the opposite position, that two high level schemas could share a lower level structure. Related to these objections is the problem that there is no good way to evaluate a system of this type. It is encouraging that the system works over some limited range of environments, but other approaches, such as production systems, also perform satisfactorily and there is no straightforward way to determine which is a "better" simulation of the human tutor.

Nonetheless the Coach system has a number of significant features. It is a well specified description of a schema-based system for interpreting complex events. Specific information, including a model of the student, is represented in a semantic network database. Generic knowledge is represented with active schemas which search for their children, modify the internal knowledge base, and perform actions in the external world. Schemas may be referred to only by their names and arguments. Schemas which do not have a parent may suggest or invoke possible parents, thus influencing their own interpretation.

Interpretation of events is based on the interaction of conceptually-guided and data-driven processing. These two processing modes interface when a schema searches for its children by examining orphaned and suggested schemas before instantiating new schemas. Errors and unexpected-events are interpreted in the same manner as correct or expected events: the corresponding schemas are activated by either conceptually-guided prediction or data-driven invocation and "interpret" lower-level schemas by incorporating acceptable ones into their structure. The level of advice given to the student derives naturally from the fact that intelligence and processing are distributed among schemas at all levels: the schema which detects an error gives advice at its own level. Thus a schema-based intelligence gives the Coach system many of the surface behaviors and underlying processes which human tutors appear to have.

### References

- Carbonell, J. R. AI in CAI: an artificial-intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems*, 1970, *MMS-11*, 190-202.
- Gentner, D. R. Toward an intelligent automated tutor. In H. F. O'Neil, Jr. (Ed.) *Procedures for Instructional Systems Development* New York: Academic Press, 1979.
- Gentner, D. R. & Norman, D. A. *The FLOW tutor: Schemas for tutoring* (Tech. Rept.). La Jolla, Calif.: University of California, San Diego, Center for Human Information Processing, 1977.
- Norman, D. A., Rumelhart, D. E., & the LNR Research Group. *Explorations in cognition*. San Francisco: Freeman, 1975.
- Raskin, J. Flow language for computer programming. *Computers and the Humanities*, 1974, 8, 231-237.
- Sleeman, D. H., & Brown, J. S. (Eds.). Intelligent tutoring systems. *International Journal of Man-Machine Studies*, 1979, *11*, 1-156.